Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Zdenko Vuk

Izdelava spletne storitve za dostop do podatkov Ubiquiti mFi naprav

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, september 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani, Zdenko Vuk, z vpisno številko **63030183**, sem avtor diplomskega dela z naslovom:

Izdelava spletne storitve za dostop do podatkov Ubiquiti mFi naprav.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Igor Rožanc,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, septembra 2014 Podpis avtorja:

Zahvala

Rad bi se zahvalil Matiji, za motiviranje in za skupaj preživete ure učenja. Rad bi se zahvalil Dinotu iz podjetja Redox d.o.o., za podporo pri diplomskem delu in za opremo. Posebna zahvala gre še mentorju viš. pred. dr. Igorju Rožancu, za njegovo razpoložljivost in pomoč.

Kazalo

Povzetek

Namen diplomske naloge je prikazati razvoj spletne storitve, ki odpravlja težave z uporabo naprav *mFi* podjetja *Ubiquiti Networks, inc.*. Težave so predvsem omejenost funkcionalnosti priložene programske opreme. Spletna storitev za komunikacijo uporablja protokol SOAP, pri katerem pa smo uporabili relativno neznan sistem za uporavljanje podatkovnih baz MongoDB.

Ubiquiti mFi je družina naprav za nadzor dogajanja v objektih. Protokol SOAP je protokol za nedvisno komunikacijo med aplikacijami. Podatkovna baza MongoDB spada v družino NoSQL podatkovnih baz. MongoVUE je aplikacija, s katero lahko upravljamo MongoDB v grafičnem uporabniškem vmesniku.

Glavne funkcionalnosti naše rešitve so neomejen dostop do podatkov *mFi* naprav iz oddaljene lokacije preko internetne povezave. Spletna storitev bo narejena z uporabo tehnologije *.NET Framework 4.5* v programskem jeziku C#. Na koncu smo delovanje rešitve preizkusili z uporabniškima aplikacijama.

Ključne besede:

Spletna storiev, naprave mFi, SOAP protokol, WSDL, MongoDB.

Abstract

The purpose of thesis is to present the development of a web service which eliminates the problems when using the *mFi* products of the company *Ubiquiti Networks, inc.*. The problems are mainly the limited functionality offered by the software bundled with these devices. The webservice uses the communication protocol SOAP. Additionally and we encountered a relatively unknown database management system named MongoDB.

Ubiquiti mFi is a family of gadgets to monitor events in buildings. The protocol SOAP is a protocol for indipendent communication between applications. The MongoDB database belongs to the family of NoSQL databases. MongoVUE offers a graphical user interface to manage the MongoDB database.

The main functionality is unlimited access to the data from the *mFi* devices over the internet from remote locations. The web service is made using the technology named *.NET framework 4.5* and written in the C# programming language. Finally, we have tested our solution with two client applications.

Key words:

Webservice, mFi devices, SOAP protocol, WSDL, MongoDB.

Poglavje 1

Uvod

V časih, ko so *pametni* telefoni tako množični in je tehnologija lažje dostopna, imajo potrošniki vse več možnosti, da upravljajo razne naprave ali pa spremljajo njihovo delovanje. Eden izmed ponudnikov takih naprav je ameriško podjetje *Ubiquiti Networks inc.*[1], ki s svojim proizvodom *mFi* ponuja možnost oddaljenega spremljanja raznih senzorjev ter naprav.
Pomanjkljivost njihovega izdelka je odsotnost praktičnega načina za oddaljen dostop do podatkovne baze, s katerim bi se lahko spremljalo podatke naprav (senzorjev). Zato morajo uporabniki uporabljati proizvajalčevo programsko opremo, ki pa ima omejene funkcionalnosti. Dve pogrešani funkcionalnosti sta recimo sproženje opozorila na uporabniški aplikaciji, če bi napravice poročale o določenih stanjih, ali obveščanje o stanju preko SMS sporočil.
V diplomski nalogi smo se lotili reševanja te pomanjkljivosti z izdelavo spletne storitve ter prikazali, kako se lahko izdela aplikacijo, ki koristi spletno storitev.
Sprva smo namestili programsko opremo za delovanje naprav na osebnem računalniku in ob tem je Ubiquiti-jeva programska oprema ustvarila tudi podatkovno bazo MongoDB. Potem smo namestili gonilnike (ang. drivers) za MongoDB ter se naučili osnovnega dela z njimi. Ko smo vzpostavili branje iz podatkovne baze MongoDB, smo se lotili raziskovanja, kako s pomočjo Visual Studia 2012 ustvariti spletno storitev, katero bi se dalo programirati v

programskem jeziku C#. Cilj je bil torej ustvariti spletno storitev, za katero bi potem lahko uporabniki ustvarili uporabniške aplikacije po svojih željah in potrebah. V spletno storitev smo dodali gonilnike za MongoDB ter pričeli dograjevati funkcije, ki jih bo spletna storitev ponujala. Na koncu smo ustvarili še dve uporabniški aplikaciji, ki preverita delovanje celote.

V naslednjem poglavju smo razložili, zakaj je rešitev, izdelana v diplomskem delu, potrebna in katere izboljšave je prinesla. Sledi še prikaz delovanja spletne storitve ter dveh aplikacij, ki kličejo funkcije na spletni storitvi ter sklepne ugotovitve.

Poglavje 2

Uporabljene tehnologije in orodja

2.1 Ubiquiti mFi

2.1.1 Splošno

Podjetje Ubiquiti Networks inc. med svojimi proizvodi (slika 2.1) ponuja tudi *mFi* [2], ki obsega ponudbo raznih senzorjev ter naprav, ki komunicirajo preko IP omrežja. Med ponujenimi napravami najdemo senzorje gibanja, temperaturo, merilnik vlage, merilnik porabe električne energije, senzor za zaznavanje odpiranja vrat in stikala z pametnim vklopkom/izklopom. Vse skupaj pa se lahko prilagodi do te mere, da lahko senzor gibanja sproži vklop luči ali pa senzor temperature sproži vklop ventilatorja.
Naprave se preko računalniškega omrežja priklopijo na računalnik, ki služi kot strežnik. Na računalniku teče proizvajalčeva aplikacija, ki spremlja delovanje naprav in zapisuje podatke v MongoDB podakovno bazo.

2.1.2 Uporabljene naprave

Za diplomsko delo smo uporabili dve napravi: napravo, ki meri temperaturo ter vlago v hiši, ter *pametni* električni razdelilec (slika 2.2). Naziv *pametni*

Slika 2.1: Predstavitvena slika iz uradne spletne strani ponudnika

.

Slika 2.2: Mpower razdelilec

.

pomeni to, da se njegov vklop nadzira na daljavo ter da zmore meriti porabo električne energije. Naprava za temperaturo in vlago se poveže v usmerjevalnik preko UTP kabla, pametni razdelilec pa preko domačega brezžičnega omrežja (ang. wifi). Obe napravi na koncu zapisujeta podatke v podatkovno bazo na osebnem računalniku, ki je tudi povezana na usmerjevalnik.

Za popolno delovanje mora biti nameščena aplikacija proizvajalca, ki, v na oko prijetnem grafičnem uporabniškem vmesniku, prikazuje dogajanje ter v ozadju zapisuje podatke v podatkovno bazo. V tej aplikaciji se lahko nastavi medsebojno delovanje naprav ter interval zapisovanja v podatkovno bazo

MongoDB.

2.2 Protokol SOAP

2.2.1 Splošno o spletnih storitvah

Spletna storitev je vmesnik do funkcionalnosti aplikacije, dosegljive preko računalniškega omrežja ter zgrajene z uporabo standardnih internetnih tehnologij (slika 2.3). Drugače povedano, če je določena aplikacija dosegljiva preko interneta, z uporabo protokolov (kot na recimo HTTP, XML, SMTP ali Jabber), potem jo imenujemo spletna storitev [3]. Neodvisna je od platforme in uporabljenega programskega jezika. Neodvisnost je ena od ključnih koristi, ki se jih pridobi z uporabo spletne storitve. Spletna storitev ne zahteva strežniškega okolja za delovanje. Lahko se ga postavi kamorkoli so na voljo standardne internetne tehnologije.
SOAP je eden izmed protokolov, ki jih za komuniciranje lahko uporabljajo spletne storitve. Bistvo SOAP-a je pošiljanje sporočil v XML formatu ter uporaba protokola HTTP [4].

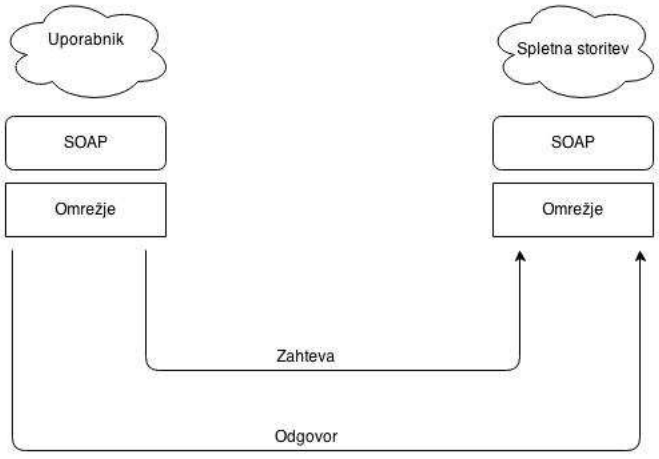
Slika 2.3: Skica spletne storitve

.

2.2.2 Spletna storitev SOAP

Prvotno je bilo ime *SOAP*, akronim za *Simple Object Access Protocol*, vendar po prihodu verzije SOAP 1.2 temu ni več tako [5]. Prvotno je SOAP

bil namenjen dostopanju do objektov, vendar je skozi čas postalo zaželjeno, da bi služil širšemu občinstvu. Zato se poudarek specifikacije hitro odmakne od predmetov k splošnim okvirom XML sporočila. Zato so specifikacijo pozneje spremenili, da ni več namenjena dostopanju do objektov, ampak do bolj splošnih XML sporočil [6].
Obstajata dva tipa SOAP zahteve: klic oddaljene procedure v RPC načinu (ang. remote procedure call) ter zahtevek po dokumentu, kjer se dokument vstavi v SOAP sporočilo in prenese do klicatelja. V diplomskem delu so bile vse zahteve prvega tipa, ker je vsaka SOAP zahteva sprožila funkcijo na spletni storitvi.
Običajno SOAP uporablja HTTP-jeva *GET* in *POST* načina pošiljanja zahtev.HTTP POST zahteva specificira vsaj dva HTTP zaglavja: *Content-Type* in *Content-Length*. Ostalih HTTP ukazov protokol SOAP ne uporablja [7].



Slika 2.4: Običajna izmenjava podatkov uporabnik - spletna storitev

.

2.2.3 Sestava sporočila

SOAP sporočilo je običajna XML datoteka, kateri pravimo tudi SOAP ovojnica (ang. envelope). Sestavljena je iz (slika 2.5)[8]:

- ovojnice (ang. envelope) - je korenski (ang. root) element XML datoteke. Ovojnica definira XML sporočilo kot SOAP sporočilo,

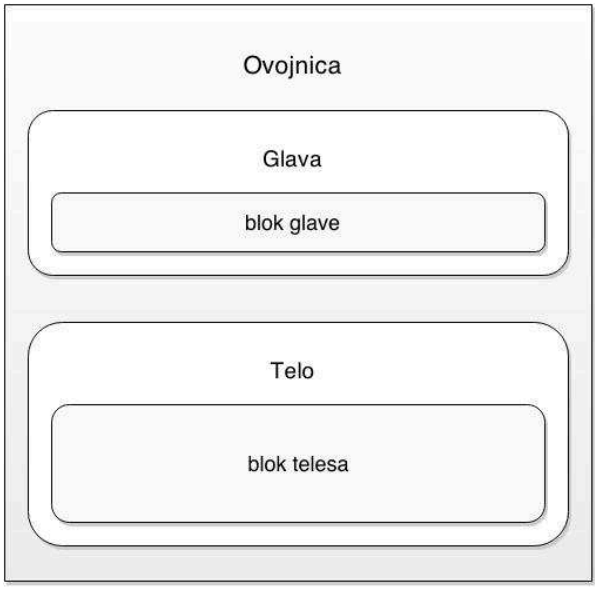
- glave (ang. header) - opcijsko - V njej najdemo podatke, ki so specifični za aplikacijo. Običajno se glava uporablja za potrebe procesiranja, avtentikacije ter usmerjanja sporočila. V glavo lahko damo tudi dodatne informacije, ki niso del glavnega dela telesa sporočila,

- telesa (ang. body) - Vsebuje dejansko SOAP sporočilo. Če sporočilo nima glave, mora bit telo prvi podelement ovojnice. Telo vsebuje SOAP zahtevek ali odgovor: ime metode ter njene parametre pri klicanju oddaljene procedure (RPC), dokument (sporočilo) ali podatke o napaki (ang. fault),

- elementa o napakah, opcijsko. Nahaja se znotraj telesa. Uporablja se za vračanje informacij o napaki pošiljatelju SOAP sporočila. Ta element se ne sme pojaviti več kot enkrat znotraj posameznega sporočila. SOAP standard definira štiri različne podelemente fault elementa: *faultcode, faultstring, faultactor, detail*.

2.2.4 Format XML

XML (ang. eXtensible mark-up language) je format za zapis podatkov [8]. Pri tem so podatki zapisani znotraj oznak (ang. tag).
Ustvarjen je bil v drugi polovici 90-ih let, njegov namen pa je bila uporaba pri prenosu podatkov. Je berljiv za človeka ali za računalnik. Za razliko od XML je bil format HTML ustvarjen za prikaz podatkov, XML pa je osredotočen na prenos.
Format je neodvisen od platforme. Sama XML datoteka ni izvršljiva, torej se je ne da zagnati ali prevesti v izvedljivo kodo.



Slika 2.5: Shema sporočila SOAP

.

2.2.5 Datoteka WSDL

WSDL (ang. Web Service Description Language) je datoteka zapisana v XML formatu, v kateri so informacije, kako koristiti spletno storitev.
Do WSDL datoteke za spletno storitev SOAP lahko dostopamo tako, da podamo njen URL in vrata ter zraven dodamo `?wsdl`. Obstajajo orodja, recimo *Visual Studio* in Javino orodje *wsimport*, katerim posredujemo naslov WSDL datoteke spletne storitve in nam orodja generirajo vse potrebno, da lahko komuniciramo s spletno storitvijo.
Datoteka WSDL vsebuje [9]:

- tip podatkov za prenašanje. Ta informacija je shranjena znotraj `types` in `message` elementov. Na sliki 2.6 so podatki tipa `string` ter `ArrayOfString`,

- tip sporočila (enosmerno ali dvosmerno), se nahaja znotraj `portType` elementa,

- informacijo o kodiranju sporočila (HTTP, HTTPS, SMTP), ki se nahaja znotraj `binding` elementa,

- končno točko (ang. endpoint) spletne storitve, znotraj *service* elementa.

.

Z uporabo datoteke WSDL lahko uporabniška aplikacija najde spletno storitev ter kliče njene funkcije. Nekaj pazljivosti je treba nameniti, če se spremeni funkcija v spletni storitvi. Takrat se WSDL datoteka lahko spremeni in posledično pride do nepravilnega delovanja uporabniške aplikacije. Lahko pa se tudi zgodi, da WSDL datoteka spremeni delovanje funkcije. Zapis WSDL datoteke v človeško berljivem XML formatu ter tudi možnost ročnega ustvarjanja WSDL datoteke, zelo olajšata delo razvijalcem, še zlasti v primeru nepravilnosti med delovanjem funkcije, WSDL opisom funkcije ali delovanjem uporabniške aplikacije.
Na sliki 2.6 vidimo primer del WSDL datoteke. Funkcija z imenom `GetExtremeValuesFromDate` sprejema podatkovni tip `string`, vrača pa tabelo nizov.

```
- <s:element name="GetExtremeValuesFromDate">
  - <s:complexType>
    - <s:sequence>
        <s:element name="dt1" type="s:string" maxOccurs="1"
          minOccurs="0"/>
      </s:sequence>
    </s:complexType>
  </s:element>
- <s:element name="GetExtremeValuesFromDateResponse">
  - <s:complexType>
    - <s:sequence>
        <s:element name="GetExtremeValuesFromDateResult"
          type="tns:ArrayOfString" maxOccurs="1" minOccurs="0"/>
      </s:sequence>
    </s:complexType>
  </s:element>
```

Slika 2.6: Del zapisa v WSDL opisu

.

2.3 Sistem za upravljanje podatkovnih baz MongoDB

2.3.1 Splošno

MongoDB je dokumentno orientiran odportokodni sistem za upravljanje podatkovne baze. Izdelalo ga je podjetje *10gen* v letu 2009. Izvorna koda je javno dostopna in jo lahko vsakdo spreminja v skladu z GNU AGPL licenco. Narejen je v programskem jeziku C++.
Dokumentno orientirana podatkovna baza pomeni, da je ustvarjena za delo z dokumentno orientiranimi informacijami [10]. Dokumentno orientirane baze podatkov so ena izmed glavnih kategorij NoSQL baz podatkov. Dokumenti v MongoDB niso vezani na določeno shemo, zato lahko podatki v določeni zbirki (ang. collection) vsebujejo različne atribute. Zaradi tega ima MongoDB dobro fleksibilnost - informacije o določenem izdelku je možno shraniti v eno zbirko, medtem ko je pri relacijskih podatkovnih bazah (ang. RDBMS) pogosta praksa, da so podatki shranjeni v več različnih tabelah.
Za uporabo teh podatkov na enem mestu, je potrebno tabele stikati (ang. join). Aplikacija določa, kateri podatki se bodo shranili v podatkovno bazo in ne obratno, kjer podatkovna baza določa, kam se bodo kateri podatki shranili.
Olajšano je tudi shranjevanje atributov objektov iz programskega jezika v podatkovno bazo, ker se lahko vse atribute shrani v en dokument. MongoDB shranjuje dokumente v zbirke, ki ne zahtevajo posebne sheme, zato se lahko dokumenti v zbirki med seboj zelo razlikujejo. To je koristno predvsem v začenih fazah izdelave aplikacij, ko se sestava dokumentov pogosto spreminja.
Dobra lastnost odsotnosti sheme je tudi ta, da lahko v zbirko dodamo dokument, ki ima na primer en dodaten atribut v primerjavi s prešnjimi (slika 2.17). Primer iz diplomskega dela je recimo dodajanje nove naprave ali senzorja, v isto zbirko kot ostale naprave, kljub temu, da ima različne atribute.

Običajno pa so si dokumenti v zbirki med seboj podobni. V zbirki, v katero mFi naprave shranjujejo podatke iz senzorjev, na primer, vsi dokumenti vsebujejo nekatere atribute.
Z uporabo MongoDB ukazne lupine, se lahko posledično vidi tudi vse informacije v človeku prijaznem JSON (JavaScript Object Notation) formatu.
MongoDB nudi tudi indeksiranje, podobno kot pri RDBMS podatkovnih bazah. Ustvari se lahko do 64 indeksov za vsako zbirko. Podpira vse tipe indeksiranja kot pri RDMBS, naraščajoče, padajoče, in celo geoprostorski (ang. geospatial) indeks.

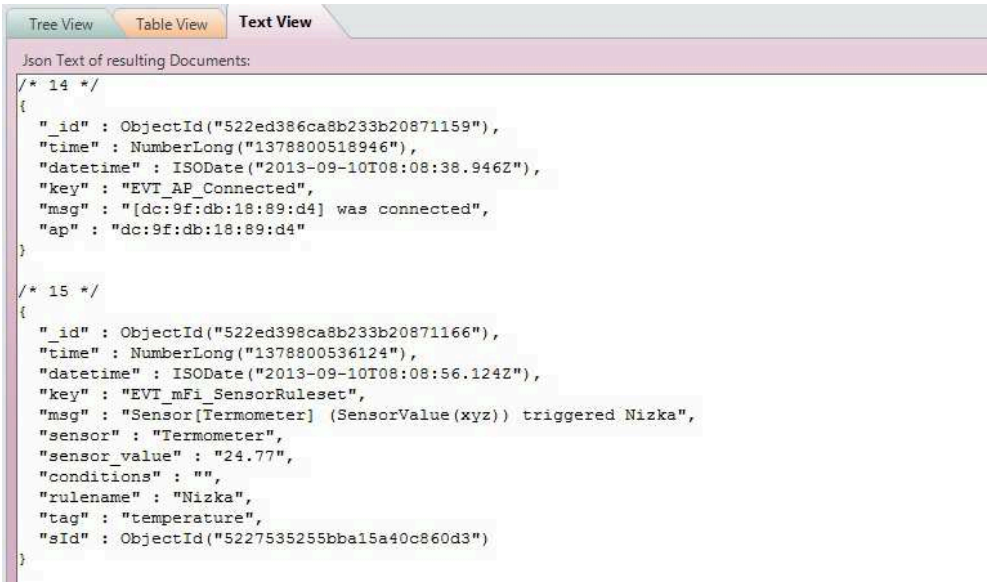
Kratek primer zapisa v MongoDB je naslednji:

```
{
   ''_id'' : ObjectId(''5227536f55bba15a40c860dc''),
   ''type'' : ''analog'',
   ''tag'' : ''temperature'',
   ''sId'' : ObjectId(''5227536f55bba15a40c860d3''),
   ''time'' : NumberLong(''1378308973000''),
   ''val'' : 26.93
}
```

MongoDB nudi odlično skalabilnost, to je možnost, da se podatkovno bazo porazdeli na več strežnikov, kar je običajen prijem za povečanje zmogljivosti podatkovne baze.

Dokumenti se vnašajo v MongoDB v obliki *ključ : vrednost*. Dokumenti v MongoDB so tipa BSON [11], torej so podobni JSON objektom v binarni obliki, kar pomeni, da so namenjeni za računalniško branje. Vrednost ključev lahko vsebuje druge dokumente, tabele in tabele dokumentov. Prednosti uporabe dokumentov so:

- podpora dokumentom v mnogih programskih jezikih,



Slika 2.7: Zaslonska slika iz MongoVUE, na kateri je razvidna odsotnost sheme

.

- sestavljeni dokumenti omogočijo, da se ne uporablja časovno potratnih združitev (ang. joins),
- uporaba različnih shem (ang. schema) omogoča polimorfizem, kar pomeni, da lahko v zbirko shranimo dokumente z različno shemo, česar nam ostale baze podatkov ne dovolijo [12].

2.3.2 Gonilniki

Splošno

Gonilniki so knjižnice v ustreznih programskih jezikih, s katerimi programski jezik dela s podatkovno bazo. Dobi se jih na uradni spletni strani. Poleg uradnih gonilnikov, obstajajo tudi gonilniki, ki jih daje na voljo skupnost. Na voljo so tudi specifikacije za izdelavo gonilnikov.

Gonilniki imajo tri glavne funkcije:

- ustvarijo ID atribut, ki je za vsak dokument enoličen,
- delajo pretvorbo v in iz BSON formata ter
- skrbijo za komuniciranje s podatkovno bazo preko TCP vtiča.

Enolična identiteta `ObjectId`

Vsak dokument v MongoDB mora imeti svoj ID atribut. V dokumentih ga najdemo pod ključem `_Id` in je tipa `ObjectId`. Sestavljen je iz dvanajstih zlogov in sicer [13]:

- UNIX časovna oznaka (ang. timestamp), to je čas v milisekundah, ki je pretekel od 1.1.1970 (4 zlogi (ang. bytes)).
- 3 zlogi *machine id*, to je enolična identiteta računalnika.
- 2 zloga *process id*, to je enolična identiteta procesa.
- 3 zlogi *counter*, torej števec zapisov v zbirko.

Teh 12 zlogov je zapisanih v šestnajstiškem formatu (dve cifri za vsak znak), zato postane zapis navidezno zelo dolg. `ID` atribut lahko ustvari tudi uporabnik.
Dobra lastnost MongoDB-jevega enoličnega identifeja je prisotnost časovnega atributa (ang. timestamp), ki ga ni potrebno beležiti drugje.
Ker se ID začne s časovnim atributom, se dokumenti urejajo v vrstnem redu po času, kar omogoča hitrejše indeksiranje po času. Časovni atribut se spremeni enkrat na sekundo. Program MongoVUE zmore prikazati čas v UTC ali UNIX formatu.
`Machine id` je ponavadi zgoščena vrednost (ang. hash) imena računalnika, na katerem je bil dokument ustvarjen. Na tak način je poskrbljeno, da različni računalniki ne tvorijo dokumentov z enakim ID-jem.
`Process id` je ustvarjen iz identifikatorja procesa (ang. process identifier, PID). To je pomembno, ker na tak način računalnik ne more ustvariti dveh

enakih ID-jev, iz dveh različnih procesov.
`Counter` je pa enostaven števec, ki poskrbi za to, da se ne ustvarjajo enaki ID-ji, na istem računalniku, v istem procesu, v času znotraj ene sekunde (vsako sekundo se časovni atribut spremeni). Na tak način je lahko za tri bajte različnih ID-jev v eni sekundi. `Timestamp` in `counter` sta shranjena po pravilu debelega konca (ang. big endian rule)[13].

JSON in BSON

JSON (ang. JavaScript Object Notation) je odprt standard za shranjevanje in pošiljanje podatkovnih objektov [14]. Izhaja iz jezika JavaScript.
Elementi v JSON so shranjeni v paru kot atribut in vrednost atributa v tekstovnem načinu. JSON podpira vse običajne tipe podatkov (števila, nize znakov, boolean vrednosti, tabele in slovarje). Večinoma je uporabljen kot alternativa XML formatu. Kot zanimivost lahko navedemo, da CouchDB shranjuje dokumente v nespremenjenem JSON formatu [15].

BSON (ang. Binary JSON) je odprt standard za zapisovanje JSON elementov v binarnem formatu [16]. Narejen je na osnovi JSON standarda. Dodali so mu podporo za podatkovne tipe `UTC datetime` (datum), `BinData` (tabele bajtov), 32 ter 64 bitni integer, števila v plavajoči vejici, `ObjectId`, del podprtih podatkovnih tipov je na sliki 2.8. Celoten seznam je dosegljiv na spletu [17]. Postopek za pretvorbo JSON v BSON uporablja tudi kodiranje, podobno TLV (ang. type-length-value), kjer se prvo določi tip podatka, potem njegova dolžina in na koncu njegova vrednost.

MongoDB uporablja datoteke v BSON formatu za naslednje stvari:

- Shranjevanje podatkov: uporabniški dokument ne sme imeti kot začetnico znak `$` ter ne sme vsebovati znaka `'.'` . Element `_id` je rezerviran kot prvotni ključ, a mu lahko uporabnik določi poljubno vrednost, če je le ta enolična (ang. unique). Gonilniki za programske jezike praviloma poskrbijo za to, da se uporabnik drži teh določil.

- Kot dokument za poizvedbo, to so dokumenti v BSON formatu, ki jih MongoDB uporablja za povpraševalne operacije.
- Kot dokument, ki vsebuje informacije, kako spremeniti dokumente v `update` operaciji [11].

```
element  ::= "\x01" e_name double                Floating point
           | "\x02" e_name string                UTF-8 string
           | "\x03" e_name document              Embedded document
           | "\x04" e_name document              Array
           | "\x05" e_name binary                Binary data
           | "\x06" e_name                       Undefined — Deprecated
           | "\x07" e_name (byte*12)             ObjectId
           | "\x08" e_name "\x00"                Boolean "false"
           | "\x08" e_name "\x01"                Boolean "true"
           | "\x09" e_name int64                 UTC datetime
           | "\x0A" e_name                       Null value
```

Slika 2.8: Nekaj podatkovnih tipov, ki jih podpira standard BSON. [17]

Primer zapisa dokumenta z vsebino `hello` v BSON formatu (`'x'`, je oznaka za šestnajstiško število):

- `4` bajti za dolžino celotnega dokumenta,
- `x00` niz, ki se konča z ničlami (ang. null-terminated string, kot v programskem jeziku C),
- `x02` za tip podatka, torej v tem primeru tipa UTF-8 string,
- `x05` je dolžina niza `hello`
- nekaj dodatnih `x00`, ker je nekaj bajtov praznih,
- `x00`, ker se niz konča z \0,
- na koncu se vsak BSON dokument zaključi z `x00`.

Pri tem velja omeniti, da BSON format uporablja pravilo tankega konca (ang. little endian)[18].

Različni viri navajajo, da format BSON prispeva k hitrosti odziva podatkovne baze, vendar zavzame tudi več prostora [19].

TCP vtičnice

TCP vtičnica je par IP naslovov in število vrat (ang. port), preko katerega poteka komunikacija.
Gonilniki komunicirajo s podatkovno bazo preko TCP vtičnice, z uporabo protokola, imenovanega *Mongo Wire Protocol* [21]. Privzeta vrata so `27017`. Ločimo akcije, kjer gonilnik čaka na strežnikov odgovor (podatkovne baze) in akcije, kjer je komunikacija s strežnikom enosmerna, torej gonilnik ne čaka na strežnikov odgovor.
Poizvedbe so poslane na strežnik, s klicanjem metode `next` na objektu kazalca (ang. cursor), strežnik potem odgovori z rezultati poizvedbe. Če je rezultatov preveč, so ti dostavljeni gonilnikom s prvim strežniškim odgovorom, čemur sledi metoda `getmore`, dokler niso dostavljeni vsi rezultati poizvedbe. Različni so primeri spreminjanja vsebine podatkovne baze (pisanje, brisanje, spreminjanje). Pri tem gonilniki pošljejo pisalni zahtevek na vtič in računajo na to, da je bilo pisanje uspešno, brez čakanja na odgovor strežnika. Odgovor ni nujno potreben, ker gonilniki sami ustvarijo `ID` atribut dokumenta in posledično ne čakajo, da strežnik ustvari ID atribut in da ga potem vrne. Če uporabnik želi potrdilo, da je bil poseg v podatkovno bazo uspešen, lahko to določi v nastavitvah, ki mu jih vsi gonilniki dajo na voljo. Pri pisanju (ang. insert) v varnem načinu (ang. safe mode) je nastavitev, pri kateri se pri pisanju čaka na odgovor strežnika, gonilnik doda ukaz `getLastError` k zahtevku za pisanje. Na tak način se gonilnik prepriča, da je sporočilo bilo dostavljeno ter da ni bilo napak pri povezavi s strežnikom. Uporabno je predvsem tedaj, ko gre za pomembne vnose v podatkovno bazo ali pa če se predvideva, da vnosi utegnejo biti neuspešni. Slabost čakanja na odgovor je zmanjšanje hitrosti.

2.3.3 Poizvedbe

MongoDB nudi na voljo lupino (ang. shell) za delo s podatkovno bazo [22]. Lupina uporablja JavaScript jezik. Najprej je treba zagnati strežniški proces `mongod`, nakar zaženemo v lupini aplikacijo `mongo`.
Za shranjevanje dokumentov se uporabljata ukaza `insert` ali `save`. Razlika med njima obstaja le v primeru, ko želimo ustvariti dokument z `Id`-jem, ki že obstaja v podatkovni bazi. V tem primeru `save` posodobi dokument, `insert` in poskusi ustvariti nov dokument ter posledično javi napako, ker dokument s tem `Id` že obstaja.
Z ukazom `count` dobimo število dokumentov v zbirki, ukaz `findOne` pa vrne en dokument, ukaz `find` pa vse dokumente (smiselno je uporabiti ukaz `limit`, s katerim omejimo število izpisov).
Na sliki 2.9 je prikazana uporaba zanke for, za generiranje dokumentov. Iz nje je tudi razvidno povečanje števca pri ID atributu ter obravnava dveh dokumentov z enako vsebino, kot dva ločena dokumenta z različnim `ID`-jem.

Z ukazom `remove` izbrišemo dokumente iz zbirke, čeprav zbirka ni čisto izbrisana, dokler njeni indeksi še obstajajo. Z ukazom `drop` odstranimo kompletno zbirko, vključno z indeksi.

2.3.4 Indeksiranje

Indeksiranje (ang. indexing) se uporablja za povečanja hitrosti poizvedb. Smiselno ga je uporabljati, ko je v zbirki veliko število dokumentov. Indekse se lahko ustvari v MongoDB-jevi lupini. Poizvedbe ustvari kazalec, s katerim se lahko premikamo (iteriramo) po rezultatih poizvedbe. Brez indeksov, bi MongoDB ob vsaki poizvedbi pregledal vse dokumente v zbirki, tako pa z indeksi naredimo urejen seznam, z vrednostjo indeksiranega atributa ter njegovo lokacijo v zbirki. Če poizvedbi dodamo ukaz `explain()`, dobimo informacijo, katere indekse je poizvedba uporabila. To nam lahko pomaga pri iskanju časovno potratnih poizvedb.
Na sliki 2.10 je razvidna uporaba ukaza `explain()`:

```
> db.m2mgeneric.findOne( { tag: "temperature", val: { $gt: 27.0 } });
{
        "_id" : ObjectId("522753c855bba15a40c860f8"),
        "type" : "analog",
        "tag" : "temperature",
        "sId" : ObjectId("5227535255bba15a40c860d3"),
        "time" : NumberLong("1378309064000"),
        "val" : 27.02
}
> db.users.insert( { username: "john" });
WriteResult({ "nInserted" : 1 })
> db.users.save( { username: "smith" });
WriteResult({ "nInserted" : 1 })
> db.users.count();
2
> db.users.find();
{ "_id" : ObjectId("538d956f0b822bcb2e996b1e"), "username" : "john" }
{ "_id" : ObjectId("538d957e0b822bcb2e996b1f"), "username" : "smith" }
> db.users.save( { username: "smith" });
WriteResult({ "nInserted" : 1 })
> db.users.find( { username: "smith" });
{ "_id" : ObjectId("538d957e0b822bcb2e996b1f"), "username" : "smith" }
{ "_id" : ObjectId("538d95b60b822bcb2e996b20"), "username" : "smith" }
> for ( var i=0; i<10; i++ ) db.numbers.insert( { num: i });
WriteResult({ "nInserted" : 1 })
> db.numbers.find();
{ "_id" : ObjectId("538d95ee0b822bcb2e996b21"), "num" : 0 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b22"), "num" : 1 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b23"), "num" : 2 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b24"), "num" : 3 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b25"), "num" : 4 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b26"), "num" : 5 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b27"), "num" : 6 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b28"), "num" : 7 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b29"), "num" : 8 }
{ "_id" : ObjectId("538d95ee0b822bcb2e996b2a"), "num" : 9 }
> db.users.remove( { username: "smith" });
WriteResult({ "nRemoved" : 2 })
> db.numbers.drop();
true
```

Slika 2.9: Primer osnovnih ukazov v mongo konzoli

- `nscanned`: vrne število dokumentov, ki so bili pregledani ob poizvedbi,
- `n`: vrne število dokumentov, ki so ustrezali iskalnim kriterijem. Velika razlika med `nscanned` ter `n` pomeni, da je poizvedba slabo optimizirana,
- `basicCursor`: osnovni kazalec, ki pove, da pri poizvedbi ni bil uporabljen noben indeks.
- `millis`: vrne čas, ki je bil potreben za poizvedbo

```
> db.m2mgeneric.find( { tag: "temperature", val: { "$gt": 28.175, "$lt": 28.5 },  } ).explain();
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 815912,
        "nscanned" : 815912,
        "nscannedObjectsAllPlans" : 815912,
        "nscannedAllPlans" : 815912,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 6374,
        "nChunkSkips" : 0,
        "millis" : 587,
        "server" : "asus1155:27017",
        "filterSet" : false
}
```

Slika 2.10: Primer uporabe ukaza `explain()`

Za tvorbo indeksa v lupini uporabimo ukaz `ensureIndex()`. Primer uporabe je na sliki 2.11.

```
> db.m2mgeneric.ensureIndex( { val: 1 } );
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 4,
        "ok" : 1
}
```

Slika 2.11: Primer uporabe ukaza `ensureIndex()`

Z ukazom `getIndexes()` dobimo spisek obstoječih indeksov. Kot je razvidno na sliki 2.12, se je z uporabo pravega indeksa zelo skrajšal čas poizvedbe.

Ukaz `stats()` prikaže razne podatke, lahko od zbirke ali od celotne podatkovne baze. Na sliki vidimo primer, ko indeksi po ključu `val` uporabljajo približno 20MB prostora na trdem disku.

Z MongoDB je mogoče tvoriti do 40 indeksov za vsako zbirko. Indeksiranje zmanjša hitrost vnosov dokumentov ter hitrost brisanja dokumentov. Vsako pisanje v podatkovno bazo, zahteva posodobitev indeksov. Zato je treba premisliti, če je pogostost branj višja kot je pogostost pisanj. Če je več pisanj kot branj, lahko indeksi zmanjšajo hitrost. Vse informacije o indeksih so shranjene v zbirki *system.indexes* v podatkovni bazi. Možno je ustvariti tudi indekse za vgrajene dokumente (ang. embedded documents).

```
> db.m2mgeneric.find( { tag: "temperature", val: { "$gt": 28.175, "$lt": 28.5 },  } ).explain();
{
        "cursor" : "BtreeCursor val_1",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 12,
        "nscanned" : 12,
        "nscannedObjectsAllPlans" : 12,
        "nscannedAllPlans" : 12,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 0,
        "indexBounds" : {
                "val" : [
                        [
                                28.175,
                                28.5
                        ]
                ]
        },
        "server" : "asus1155:27017",
        "filterSet" : false
}
```

Slika 2.12: Skrajšan čas poizvedbe na manj kot eno milisekundo

MongoDB pozna tudi sestavljene indekse (ang. compound indexes), kjer indeksiramo dva ali več atributov, recimo po ključu `tag`, potem pa še po ključu `val` - kot je prikazano na sliki 2.13.

```
> db.m2mgeneric.ensureIndex( {"tag": 1, "val": 1 });
```

Slika 2.13: Primer sestavljenega indeksa

2.3.5 GridFS specifikacija

Največja velikost dokumenta v MonguDB je štiri megabajte. Zato se večje dokumente shranjuje po GridFS specifikaciji [21], po kateri se dokumenti razbijejo na več manjših delov. Prednost tega je, da iskanje dela dokumenta ne zahteva branje celotnega dokumenta. Vsi MongoDB-jevi gonilniki uporabljajo GridFS, za shranjevanje večjih dokumentov.

GridFS nima vgrajenega protokola za zaklepanje ob spreminjanju, zaradi istočasnega dostopa do dokumenta iz več procesov. Zato je uporaba

GridFS omejena na `put, get` in `delete` operacije. Posledično se ob posodobitvi dokumenta zahteva najprej izbrisanje, potem pa tvorbo novega. Natančneje rečeno, GridFS dokument razdeli v več 256kB velikih delov , potem shrani vsakega od teh kot posamezni dokument. Pri tem pomaga sposobnost MongoDB-ja shranjevanja podatkov v binarnem formatu (BSON). Taki dokumenti so privzeto shranjeni v zbirki z imenom `fs.chunks`. Ko so delčki shranjeni, so tudi datotekini meta podatki (ime datoteke, velikost, itd) shranjeni kot dokument v zbirko `fs.files`. Poleg gonilnikov, zmore souporabljati z GridFS tudi `mongotools`. GridFS najbolje deluje z velikimi datotekami, ki se redkokdaj spreminjajo.

2.4 Ostala orodja

2.4.1 Orodje Java wsimport

Wsimport je orodje, kateremu podamo naslov WSDL datoteke in nam ustvari javanske razrede (ang. stub classes), za delo s spletnimi storitvami [20]. Wsimport dobimo pri namestitvi javanskega JDK (Java development kit). V Windows okolju zaženemo `wsimport.exe`, v ukaznem oknu ter mu kot argument določimo povezavo do datoteke spletne storitve. Generirane razrede uporabimo med izdelavo odjemalca.

2.4.2 MongoVUE

MongoVUE je program, s katerim lahko upravljamo z `mongodb` bazo podatkov v grafičnem uporabniškem načinu (slika 2.15). Omogoča nam, da se izognemo uporabi lupine (ang. terminal). Na tak način se ni treba poglabljati v posebnosti ukazov, ki jih razume `mongodb`. Med prednostmi tega orodja je tudi priročnejši prikaz vsebine podatkovne baze. Dodatna funkcionalnost MongoVUE je pomoč pri učenju ukazov, za uporabo MongoDB-jeve ukazne lupine (katera uporablja jezik JavaScript): medtem, ko uporabnik uporablja

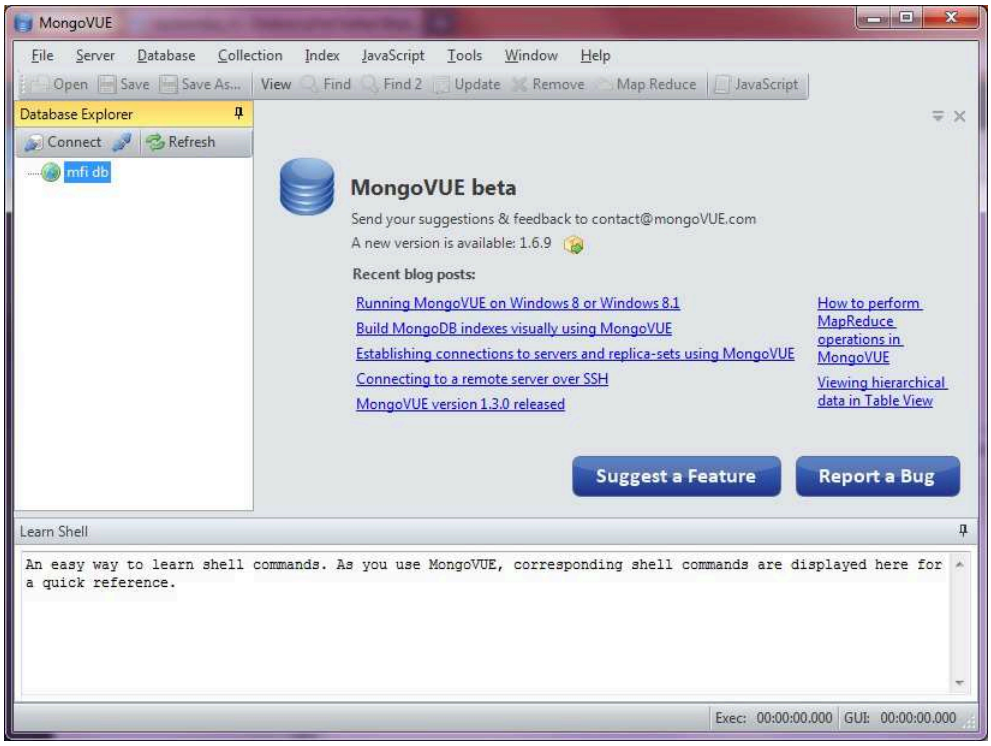
```
C:\Program Files\Java\jdk1.7.0_02\bin>wsimport.exe -keep -verbose -d C:/test htt
p://localhost:45702/Service1.asmx?wsdl
parsing WSDL...


[WARNING] Ignoring SOAP port "Service1Soap12": it uses non-standard SOAP 1.2 bin
ding.
You must specify the "-extension" option to use this binding.
  line 309 of http://localhost:45702/Service1.asmx?wsdl


Generating code...

org\tempuri\ArrayOfString.java
org\tempuri\GetExtremeValuesEver.java
org\tempuri\GetExtremeValuesEverResponse.java
org\tempuri\GetExtremeValuesFromDate.java
org\tempuri\GetExtremeValuesFromDateResponse.java
org\tempuri\GetExtremeValuesToday.java
org\tempuri\GetExtremeValuesTodayResponse.java
org\tempuri\GetTagFromDatetime.java
org\tempuri\GetTagFromDatetimeResponse.java
org\tempuri\NumberOfEntriesEventsThread.java
org\tempuri\NumberOfEntriesEventsThreadResponse.java
org\tempuri\ObjectFactory.java
org\tempuri\ReadLastNEventsThread.java
org\tempuri\ReadLastNEventsThreadResponse.java
org\tempuri\ReadLastValuesThread.java
org\tempuri\ReadLastValuesThreadResponse.java
org\tempuri\Service1.java
org\tempuri\Service1Soap.java
org\tempuri\package-info.java

Compiling code...

javac -d C:\test -classpath C:\Program Files\Java\jdk1.7.0_02/lib/tools.jar;C:\P
rogram Files\Java\jdk1.7.0_02/classes -Xbootclasspath/p:C:\Program Files\Java\jd
k1.7.0_02\jre\lib\rt.jar;C:\Program Files\Java\jdk1.7.0_02\jre\lib\rt.jar C:\tes
t\org\tempuri\ArrayOfString.java C:\test\org\tempuri\GetExtremeValuesEver.java C
:\test\org\tempuri\GetExtremeValuesEverResponse.java C:\test\org\tempuri\GetExtr
emeValuesFromDate.java C:\test\org\tempuri\GetExtremeValuesFromDateResponse.java
 C:\test\org\tempuri\GetExtremeValuesToday.java C:\test\org\tempuri\GetExtremeVa
luesTodayResponse.java C:\test\org\tempuri\GetTagFromDatetime.java C:\test\org\t
empuri\GetTagFromDatetimeResponse.java C:\test\org\tempuri\NumberOfEntriesEvents
Thread.java C:\test\org\tempuri\NumberOfEntriesEventsThreadResponse.java C:\test
\org\tempuri\ObjectFactory.java C:\test\org\tempuri\ReadLastNEventsThread.java C
:\test\org\tempuri\ReadLastNEventsThreadResponse.java C:\test\org\tempuri\ReadLa
stValuesThread.java C:\test\org\tempuri\ReadLastValuesThreadResponse.java C:\tes
t\org\tempuri\Service1.java C:\test\org\tempuri\Service1Soap.java C:\test\org\te
mpuri\package-info.java
```

Slika 2.14: Primer dela z wsimport

MongoVUE, se ekvivalentni lupinski ukazi prikazujejo v posebnem oknu.

Za potrebe diplomskega dela smo uporabljali brezplačno (obstaja tudi plačljiva) različico, ki je zadoščala našim potrebam, preden smo razumeli, kako mFi vpisuje podatke v opdatkovno bazo, po katerem vrstnem redu so podatki vpisani, katere atribute imajo razni vnosi ter imena ustvarjenih zbirk. Za dostop do podatkovne baze iz MongoVUE potrebujemo naslov strežnika ter vrata, na katerih posluša `mongodb` strežnik (ang. mongodb server (`mongod`)). Privzeta vrata so `27017`, ker pa ta vrata že uporablja mFi, smo izbrali vrata `37017` (slika 2.16).
Na sliki 2.17 je primer, kakšne podatke mFi zapiše v podatkovno bazo.

Opisali smo Javino orodje wsimport in prikazali njegovo uporabo. Nato

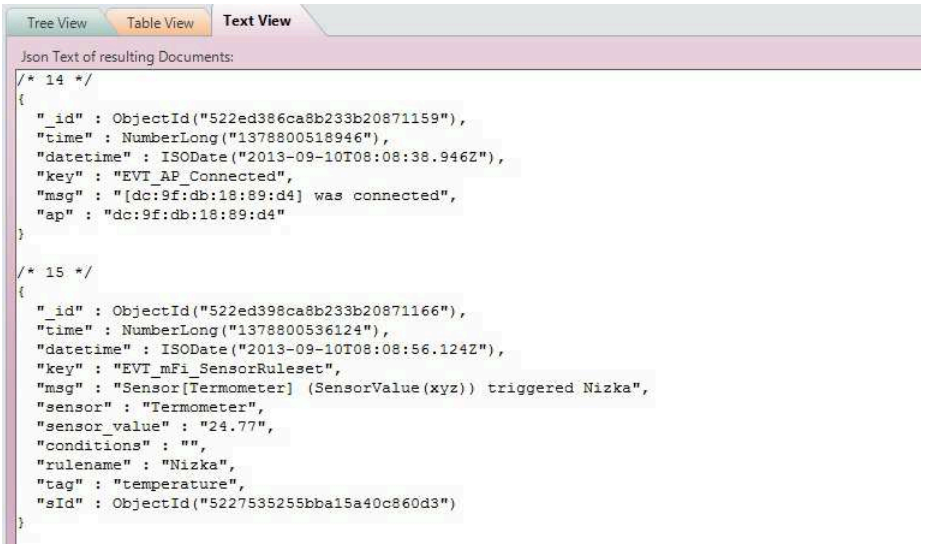


Slika 2.15: Zajem zaslona iz MongoVUE

smo še predstavili orodje MongoVUE in pojasnili, zakaj nam je bilo to orodje koristno.

Sledi definicija problema diplomske naloge; to je motivacija, zakaj je v diplomski nalogi prikazana rešitev koristna.

Slika 2.16: Nastavitve povezave do podatkovne baze

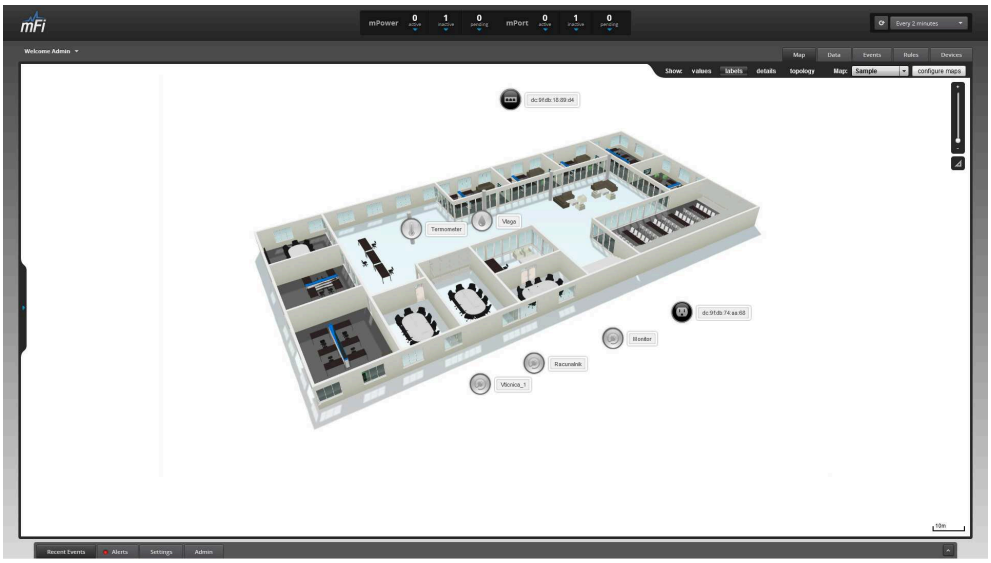


Slika 2.17: Primer vnosa podatkov naprave v database

Poglavje 3

Opis problema

mFi-jeva programska oprema, ki jo pridobimo poleg njihovih naprav, prejema podatke od naprav ter jih shranjuje v podatkovno bazo MongoDB na računalniku, ki za to nalogo deluje kot strežnik. Naš cilj je imeti vpogled v dogajanje v podatkovni bazi iz oddaljene lokacije, da bi tako lahko izvedeli, kakšno je stanje na objektu, konkretno temperatura, stopnja vlage ter poraba električne energije treh naprav.
Ker proizvajalčeva programska oprema ne nudi vseh želenih funkcionalnosti, je smiselno ustvariti spletno storitev, s katero bi lahko vgradili dodatne funkcionalnosti.
Proizvajalec naprav sicer ponuja tudi možnost oddaljenega dostopa do aplikacije, vendar pomanjkljivost omejenega obsega funkcionalnosti ostane. Pri tem nas ovira še dejstvo, da so podatki shranjeni v NoSQL podatkovni bazi in ne moremo ubrati bližnjic za oddaljen dostop do podatkovne baze, kot bi jih lahko z uporabo MySQL. Zato smo se odločili za izdelavo spletne storitve, ki dostopa do podatkovne baze ter za komunikacijo uporablja SOAP protokol.



Slika 3.1: Zaslonska slika mFi-jeve aplikacije

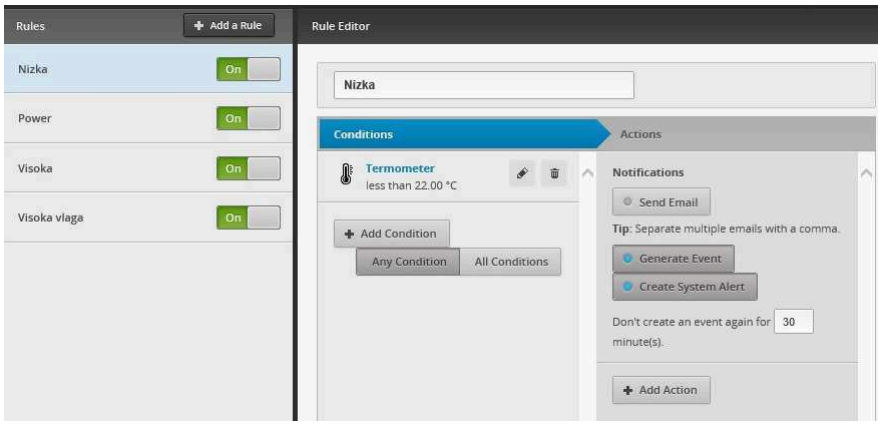
3.1 Obstoječa rešitev

V *mFi*-jevi aplikaciji ima uporabnik možnost določiti dogodke (ang. events); to so akcije, ki se sprožijo, ko naprave sporočijo določene vrednosti. Te vrednosti določi uporabnik sam, na primer za visoko temperaturo, zaznano gibanje, zaznano odpiranje vrat ... Ti dogodki se vpišejo v MongoDB v `event` zbirko. Možnosti za obveščanje uporabnika o teh dogodkih so v tej aplikaciji omejene na dogodke na računalniku ter na obvestilo preko email sporočila. Tukaj manjkajo alternativne možnosti, da bi lahko bil uporabnik obveščen na daljavo.

3.2 Uporabniške zahteve

Funkcionalnosti, ki jih bo naša spletna storitev ponujala, so:

- neomejen dostop do kakršnihkoli podatkov iz podatkovne baze,
- možnost ustvarjanja alarma (zvočni, SMS, obveščanje ipd) na uporabniški aplikaciji, ki dostopa do spletne storitve,



Slika 3.2: Nastavitev opozoril

- možnost spreminjanja prejetih podatkov, na primer popravilo časa ob sezonskem zamiku ure,
- zbiranje statističnih podatkov,
- neodvisnost, ki jo dobimo z uporabo SOAP protokola,
- uporaba standardiziranih protokolov,
- uporaba običajne cenovno ugodne internetne povezave,
- prejemanje in pošiljanje podatkov tipa `string` ter
- delovanje na domačem računalniku z operacijskim sistemom *Windows*.

Naša spletna storitev bo vsebovala le bralne dostope do podatkovne baze, zato je strah, da bi namerno ali nenamerno prišlo do spremembe podatkov v podatkovni bazi, odveč. Ker posredovani podatki spletne storitve niso zaupne narave, nam ni bilo treba posvečati pretirane pozornosti za omejevanje dostopa do spletne storitve.

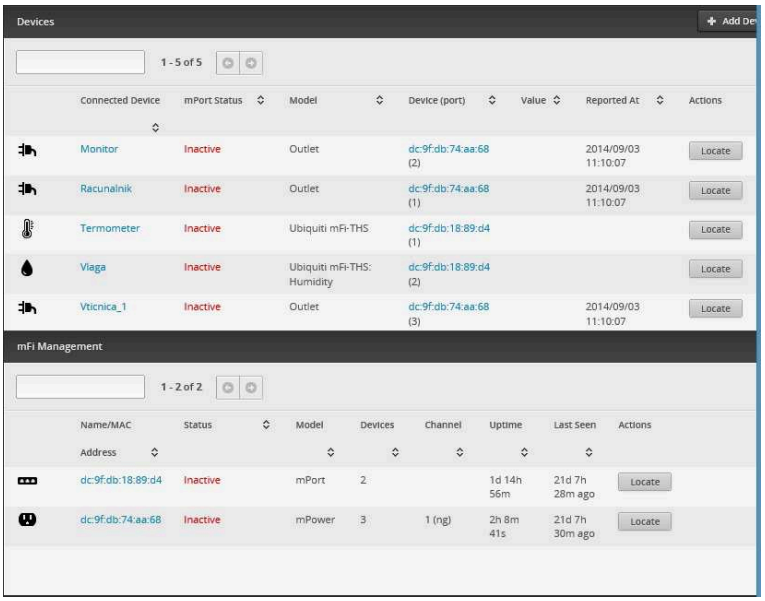
Naloga naše spletne storitve je posredovati vse za končnega uporabnika koristne informacije iz podatkovne baze. Zato atributa `type`, ki ima v našem primeru vedno vrednost `analog`, nikoli ne bere. Ustvarjene funkcije naj bi nam lahko posredovale:

Key	Value	Type
(15) {..}		Document
_id	522ed398ca8b233b20871166	ObjectId
time	1378800536124	Int64
datetime	10.9.2013 8:08:56	DateTime
key	EVT_mFi_SensorRuleset	String
msg	Sensor[Termometer] (SensorValue(xyz)) triggered Nizka	String
sensor	Termometer	String
sensor_value	24.77	String
conditions		String
rulename	Nizka	String
tag	temperature	String
sId	5227535255bba15a40c860d3	ObjectId

Slika 3.3: Primer vnosa v `event` zbirko

- mejne vrednosti senzorjev. To je koristno, ker nudi informacije, če je prevroče/premrzlo v objektu ter do koliko električne energije so porabljali priključki na *mpower*. Pri mejnih vrednostih je pomembno, da dobimo podatke o minimalni in maksimalni vrednosti,
- vrednosti senzorjev v določenem času,
- najbolj nedavne vrednosti senzorjev,
- podatek, ali je prišlo do prekoračitve alarmne meje senzorjev. V primeru prekoračitve se dogodek zapiše v `event` zbirko. Ker so v tej zbirki prisotni tudi stari podatki, je potrebno izumiti način, kako preveriti, če so prisotni tudi nedavno narejeni vnosi v to zbirko (primer vnosa) 3.3.

Na sliki 3.4 so prikazani podatki naprav, ki smo jih uporabljali. Namestitev naprav deluje na naslednji način: *mFi-THS* (senzor za temperaturo in vlago) je preko UTP/RJ45 kabla povezan na *mPort*, slednji je na enak način povezan v usmerjevalnik. *MPower* (*pametna* vtičnica, ki zmore meriti porabo električne energije), je povezana na usmerjevalnik preko hišnega brezžičnega omrežja. Računalnik, na katerem teče *mFi*-jeva aplikacija, je seveda tudi povezan z usmerjevalnikom.
Aplikacija se lahko namesti na računalnik, na katerem je nameščen *Microsoft Windows* ali Linux (priporočen je *Ubuntu*), Java JRE in FlashPlayer 10. Na sliki 3.1 je glavna stran aplikacije.



Slika 3.4: Naprave, ki bi lahko uporabljali med izdelavo diplomskega dela

Poglavje 4

Tehnična rešitev

4.1 Ideja

Spletna storitev bo narejena v programskem jeziku C# z orodjem Visual Studio 2012. Delovala bo na računalniku, na katerem deluje tudi programska oprema *mFi* naprav (in z njo podatkovna baza MongoDB). Za dostop do MongoDB bo uporabljala MongoDB-jeve gonilnike za programski jezik C#. Za preizkus delovanja spletne storitve bomo naredili dve aplikaciji.
Prva bo dostopala do spletne storitve iz konzolnega programa narejenega v programskem jeziku Java in bo služila le kot potrdilo, da se z uporabo orodij hitro dobi povezavo do spletne storitve ter da le-ta zanesljivo deluje v različnih programskih jezikih. Druga bo okenski program, narejen v programskem jeziku C# z .NET tehnologijo WPF, s katero lahko pošiljamo zahtevke spletni storitvi preko gumbov ter dobimo izpisane rezultate v WPF-jevski TextBox.

4.2 Spletna storitev

Spletna storitev privzeto teče na vratih `45702` in na ta vrata se povežejo uporabniške aplikacije. Možno je poljubno spremeniti vrata, vendar zahteva tudi posodobitev nastavitev vrat v uporabniških aplikacijah. Za komunika-

cijo uporablja protokol SOAP, kar pomeni protokol HTTP in v nižjih plasteh protokol TCP/IP. Iz naslednje vrstice v WSDL datoteki spletnega servisa:

```
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
```

je razvidno, da uporablja SOAP različico 1.2. (`soap12`).
Ob vsakem klicu metode na spletni storitvi se ustvari nova nit. Nit nato čaka določen čas na odgovor klicane funkcije. Nekatere funkcije se sprehodijo po veliki količini podatkov iz podatkovne baze, kar utegne zahtevati precej časa. Tiste funkcije, ki berejo relativno sveže podatke, zmorejo dokončati operacijo hitreje, zato tudi nit, ki jo kliče, čaka manj časa. Če odgovora v tem času ne dobi, kliče funkcijo `Abort()` in vrne uporabniku sporočilo o napaki. Spodaj je primer niti, ki kliče funkcijo in na njen odgovor čaka 20 sekund (metoda `Join`), ter kratkega komentarja delovanja funkcije (`WebMethod(Description)`), ki je viden tudi v WSDL datoteki:

```
[WebMethod(Description="Returns the highest and lowest values for
each sensor. However it does not count (very) old entries in data
base. Returned array: {SId, date time, value}")]
public string[] GetExtremeValuesEver()
{
    string[] res = String.Empty;
    Thread th1 = new Thread(() => { res = GetExtremeValuesEverOnS
    erver();
    });
    th1.Start();
    th1.Join(20000);
    return res;
}
```

Pri klicanju funkcij uporabljamo zaklep (`mutex`), ki poskrbi, da dve ali več niti hkrati ne dostopata do vira, v tem primeru do podatkovne baze [23].

Zaklep omogoča dostop do vira le eni niti hkrati. Ko funkcija konča branje iz podatkovne baze, sprosti zaklep. Če pride pri branju podatkov iz podatkovne baze do napake, lahko pride do tega, da je zaklep še vedno aktiven, zato je treba poskrbeti, da se ob napaki funkcije zaklep vseeno sprosti. Seznam funkcij se lahko v prijetni obliki vidi v spletnem brskalniku, če se v naslov do spletne storitve doda `?wsdl`.

Funkcije spletne storitve so zasnovane tako, da se lahko naknadno doda nove *mFi* naprave. Če se doda naprave, recimo senzor temperature in vlage ali napravo za merjenje porabe električne energije, bo predvidoma delovalo brezhibno. Če bi se dodalo naprave z zelo različnimi vnosi od obstoječih (recimo senzor za odpra vrata z vrednostmi `true/false`), bi bilo potrebno prilagoditi funkcije. Če ostale naprave vrednosti senzorjev vnesejo pod atribut `value`, bi bilo dovolj spremeniti funkcije tako, da vračajo tudi `SId` atribut, potem ostalo prepustiti uporabniški aplikaciji. Spletna storitev namreč loči naprave po njihovem `SId` ključu (`Sensor Id`). Spisek naprav se pri branju podatkov shranjuje v seznam. Vsakič, ko pri branju naleti na `SId`, ga doda. Zavoljo hitrosti pri tem se funkcija ne sprehodi po celotni zbirki v podatkovni bazi, ampak bere le do prednastavljene globine (`lengthReduced`). `lengthReduced` je celoštevilska spremenjlivka, ki določa, kako globoko naj se funkcija sprehodi po zbirki. Ker se je vnosov nabralo že milijon (in še več bi jih lahko bilo ob redni uporabi), bi bilo sprehajanje po celotni zbirki preveč časovno potratno.
Spodaj je primer, kako z `lengthReduced` omejimo število iteracij v zanki. Razvidno je tudi, kako v zanki ustvarjamo seznam naprav (razlikujejo se po `SId`):

```
List<ObjectId>SIdList = new List<ObjectId>();
for(int i=length-1; i>=0 && i>(lengthReduced); i--)
{
    ObjectId tempSId = reducedList[i].SId;
    if(!SIdList.Contains(tempSId))
```

```
    {
        SIdList.Add(tempSId);
    }
}
```

Spletna storitev vrača podatke tipa `string` ter kot tabelo nizov (ang. string array). Drugih podatkovnih tipov zaradi zanesljivosti delovanja ne uporablja. Možno bi bil sicer tudi vračati podatke kot objekte, narejene po meri ali na primer kot objekt tipa DateTime, vendar lahko pride do nevšečnosti, če uporabniška aplikacija ne pozna (ali pa drugače dojema) vrnjenega tipa podatka. Podatke tipa `string` pa pravilno dojemajo vse uporabniške aplikacije.

Spletna storiev se poveže na podatkovno bazo preko vrat `37017`, ker so vrata `27017` že zasedena (uporabljajo jih *mFi*-jeve naprave). V delovanju morajo naprave redno posredovati podatke, tako da jih lahko z uporabniškimi aplikacijami spremljamo.

4.3 Dostop do podatkovne baze MongoDB

Pri dostopu do podatkovne baze imajo glavno vlogo MongoDB gonilniki. Z njimi dobimo potrebne razrede, objekte in metode, da lahko dostopamo do podatkovne baze. Referenco do MongoDB gonilnikov dobimo z vnosom

```
using MongoDB.Bson;
using MongoDB.Driver;.
```

Vključene so bile še naslednje MongoDB-jeve knjižnice:

```
MongoDB.Driver.Builders;
MongoDB.Driver.GridFS;
MongoDB.Driver.Linq;.
```

Vseh sedem ustvarjenih funkcij bere podatke iz podatkovne baze in vrača rezultat kot tip `string` ali kot tabelo nizov. Pri iskanju določenih zapisov začne z branjem na koncu zbirke, kjer so najbolj sveži vnosi. Vnosi v MongoDB so že v začetku sortirani po času. Ker je lahko zapisov veliko, je smiselno omejiti globino iskanja po zbirki.

Senzor temperature in vlage ima naslednjo obliko zapisa v MongoDB:

```
''_id'' :  ObjectId(''5227536f55bba15a40c860dc''),
''type'' :  ''analog'',
''tag'' :  ''temperature'',
''sId'' :  ObjectId(''5227536f55bba15a40c860d3''),
''time'' :  NumberLong(''1378308973000''),
''val'' :  26.93
```

Pri tem nam `sId` pove, za kateri senzor gre, `tag` nam pove, če gre za meritev temperature ali vlage, `time` je čas v UNIX formatu, `val` pa je vrednost. Merilec porabe električne energije ima podobno strukturo, v primeru priklapljanja novih naprav, pa bi bilo potrebno preveriti, kakšne zapise delajo naprave v MongoDB (katere pare `ključ:vrednost`).

Kot je priporočeno v navodilih za MongoDB smo zgradili svoje domenske razrede: `MongoEntity` vsebuje le `ObjectId` atribut (prikazano spodaj).

```
[BsonId]
public ObjectId Id { get; set; }
```

Razreda `MongoEvent` in `MongoM2MGeneric` dedujeta od `MongoEntity` ter definirata konstruktorje za preostale ključe v podatkovni bazi (primer spodaj za atribut `Time`).

```
[BsonElement("time")]
public Int64 Time { get; set; }
```

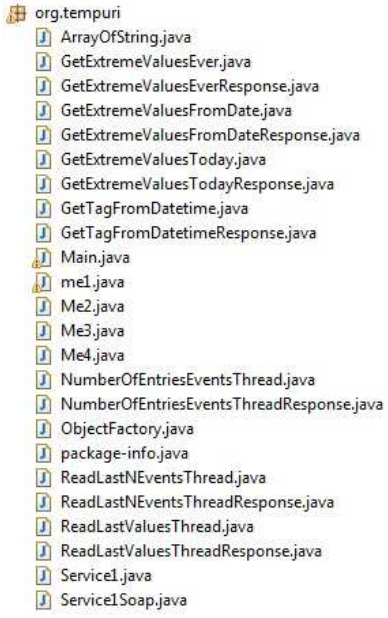
4.4 Opis obeh aplikacij

Za prikaz delovanja smo ustvarili dve aplikaciji. Prva je osnovna konzolna aplikacija v Javi, ki služi samo potrjevanju, da se z uporabo orodij hitro dobi povezavo do spletne storitve ter da le-ta zanesljivo deluje v različnih programskih jezikih.

Druga je okenski program, narejen v C# z .NET tehnologijo WPF, s katerim lahko pošiljamo zahtevke spletni storitvi preko gumbov ter dobimo izpisane rezultate v WPF-jevski TextBox.

4.4.1 Konzolna aplikacija v Javi

Aplikacijo smo izdelali z programom Eclipse, bolj natančno z različico IDE for Java EE Developers. Po dodani referenci do lokacije `wsdl` datoteke spletne storitve (kjer se je uporabljal paket JAX-WS), smo ročno premaknili v razvojno okolje razrede, ki smo jih pridobili z uporabo orodja wsimport. Funkcije spletne storitve kličemo z istoimensko funkcijo, kjer je prva črka pretvorjena v malo (primer spodaj).



Slika 4.1: Razredi v javanski aplikaciji

```
String s=service.getService1Soap().
numberOfEntriesEventsThread();
```

Z uporabo `Timer` in `TimerTask` dosežemo periodično klicanje funkcije.

4.4.2 Okenski program

Med razvojem programa je potrebno poznati URI spletne storitve, ker to informacijo potrebujemo za dodati referenco do storitve (ang. service reference). Na tak način dobimo dostop do spletne storitve. V nastavitvah lahko uporabnik poimenuje naprave, kjer se določijo pari `SId` - ime. Imena se uporabljajo za prikaz v aplikaciji. Te nastavitve se shranijo v datoteko na računalniku, potem pa jih aplikacija prebere ob vsakem zagonu.

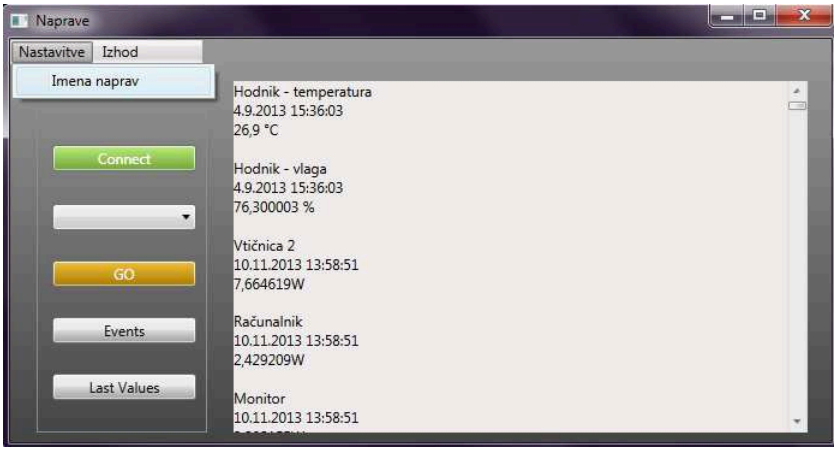
Kratek opis delovanja programa:

- Z gumbom *Connect* se sproži klic funkcije `System.Reflection.MethodInfo[] methods = se1.GetType().GetMethods();` s katero dobimo dostop do obstoječih funkcij spletne storitve, katere lahko izberemo v *ComboBox* (element pod gumbom Connect). Z gumbom *GO* kličemo funkcijo, izbrano v *ComboBox*-u. Pri tem velja omeniti, da sta spodnja dva gumba funkcionalna tudi brez klika na zgornje gumbe, saj program dobi podatke (in s tem funkcije) ob dodajanju reference na spletno storitev.

- Z gumbom *Events* dobimo dežuranje nad dogodki. Z uporabo metod razreda `Timer` vsakih pet minut povprašamo spletno storitev za število vnosov v `event` zbirki.

```
Timer myTimer = new Timer();
myTimer.Elapsed += new ElapsedEventHandler(NumberEvents);
myTimer.Interval = 300000; //5 minut
myTimer.Start();
```

  Za posodabljanje grafičnega vmesnika (ang. UI) v WPF izven grafičnovmesnikove niti si pomagamo z `Dispatcher.CheckAccess` metodo, ker načeloma lahko grafični vmesnik posodablja le nit, ki ga vodi. Spodaj je prikazano tako posodabljanje vsebine v `TextBlock`:



Slika 4.2: Zaslonska slika okenskega programa

```
if (textBlock1.Dispatcher.CheckAccess())
{
    textBlock1.Text += izpis;
}
else
{
    Action act = () => { textBlock1.Text += izpis; };
    textBlock1.Dispatcher.Invoke(act);
}
```

Za proženje zvočnega signala ob novih dogodkih smo si pomagali z razredom `SoundPlayer`:

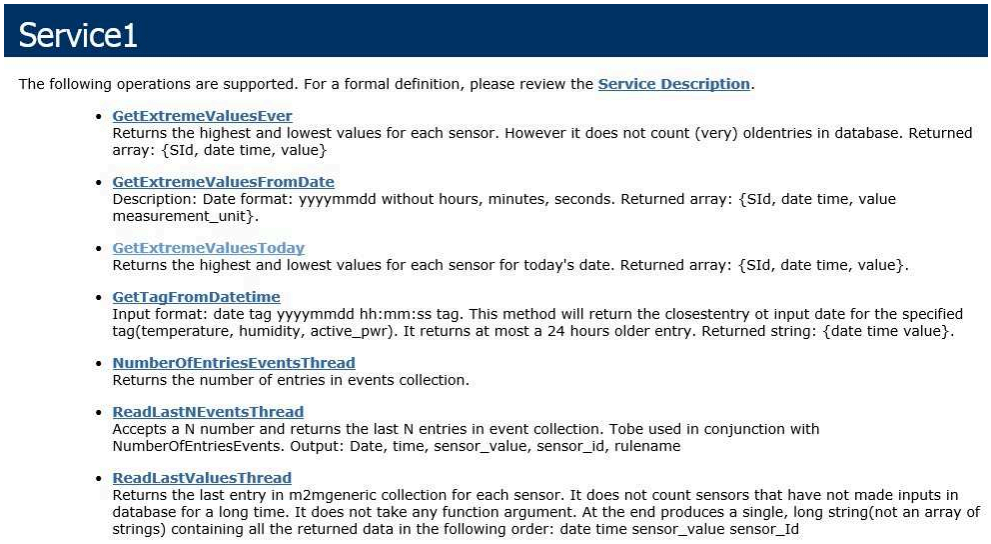
```
System.Media.SoundPlayer play = new System.Media.SoundPlaye
r(@"C:\zvoki\dogodek.wav");
```

- S klikom na gumb *LastValues* dobimo zadnje vrednosti prisotnih naprav.

4.5 Podroben prikaz tehnične rešitve

Predstavljene bodo funkcije, ki jih spletna storitva nudi. Nato bo prikazan glavni del izvorne kode razreda, kjer je med drugim z uporabo razredov `mongoClient` in `mongoServer` definirana povezava do podatkovne baze. V nadaljevanju je prikazano preverjanje pravilnosti prejetih argumentov ene izmed funkcij. Sledi še primer poizvedbe v MongoDB iz programskega jezika C#. Na koncu je še prikazana obravnava izjem pri eni izmed funkcij.

Ob spoznanju, da je najmanj možnosti za napake, če spletna storitev vrača enostavnejše podatkovne tipe, smo vse funkcije popravili tako, da vračajo rezultat bodisi tipa `string` bodisi kot tabelo nizov.
Ob klicu vsake funkcije se sproži nova nit na strežniku, tako da spletna storitev preverjeno lahko streže več zahtevam hkrati.
Seznam ustvarjenih funkcij je na sliki 4.3.



Slika 4.3: Seznam ustvarjenih funkcij

Kratek opis funkcij:

- `GetExtremeValuesEver`: vrača največjo in najnižjo vrednost, z vsako napravo posebej (za vsak `SId`). Ker bi branje celotne zbirke lahko

trajalo preveč časa je globina branja omejena s spremenljivko na nivoju razreda (njena vrednost je `1000000`). Rezultat vrne kot enodimenzijsko polje, v katerem je za vsako napravo šest polj: tri polja (`SId`, čas in vrednost) za oba primera minimalne in maksimalne vrednosti.

- `GetExtremeValuesFromDate`: sprejema en argument in sicer niz kot datum v formatu `yyyymmdd`. Nato datum pretvori v objekt tipa `DateTime` in iz slednjega v `Int64`. Podobno kot zgornja funkcija vrne enodimenzijsko polje, v katerem je za vsako napravo šest polj, z razliko, da vrne vrednosti za določen datum med 00.00 in 23.59 uro.

- `GetExtremeValuesToday`: podobno kot zgornja funkcija, vrne vrednosti za današnji dan od 00.00 do trenutnega časa.

- `GetTagFromDatetime`: v argumentu ji podamo datum, uro in tip senzorja, vrne nam vrednost senzorjev željenega tipa (temperatura, vlaga, poraba el. energije) v določenem času. Ker zapisa v zbirki za točno določen čas verjetno ni (zapisi se privzeto izvajajo s periodo dveh minut), funkcija išče do 24 ur novejše vnose.

- `NumberOfEntriesEventsThread`: vrne število vnosov v `event` zbirki, kateri so bili vnešeni zaradi preseganja vrednosti senzorjev, ki jih je določil uporabnik v *mFi* programski opremi. Med branjem vnosov ignoriramo tiste, ki jih niso sprožila obvestila.

  ```
  list.RemoveAll(item => item.Key != "EVT_mFi_SensorRuleset")
  ```

  S primerjavo odgovora s prejšnjim, dobimo podatek, koliko novih alarmnih dogodkov se je zgodilo med tem časom.

- `ReadLastNEventsThread`: ko z uporabo zgornje funkcije dobimo število novih dogodkov, to število podamo tej funkciji kot argument in nam vrne informacije (datum, čas, vrednost senzorja, `SId` senzorja ter naziv dogodka) o zadnjih `N` vnosih v `event` zbirko, kjer število `N` določi uporabnik.

- `ReadLastValuesThread` vrne zadnji vnos v zbirko za vsak senzor posebej

Spodaj je prikazan glavni del razreda `ConnectionToDatabase` v katerem je definirana povezava do MongoDB:

```
public class ConnectionToDatabase
{
    string connectionString;
    MongoClient mongoClient;
    MongoServer mongoServer;
    public string database;
    public MongoDatabase db;

    try
    {
        connectionString = "mongodb://localhost:37017";
        mongoClient = new MongoClient(connectionString);
        mongoServer = mongoClient.GetServer();
        database = "ace";
        db = mongoServer.GetDatabase(database);
    }
    catch (Exception e)
    {
        throw new Exception(e.ToString());
    }
}
```

Spodaj je prikazana izbira zbirke `m2mgeneric`, v kateri so shranjeni redni podatki iz senzorjev, pri čemur je spremenljivka `ctd1` objekt tipa `ConnectionToDatabase`.

```
MongoCollection<MongoM2MGeneric>data =
```

```
ctd1.db.GetCollection<MongoM2MGeneric>
("m2mgeneric");
MongoCursor<MongoM2MGeneric>
allDocs = data.FindAllAs<MongoM2MGeneric>();
```

Funkcije, ki sprejemajo argument, imajo preverjanje pravilne oblike argumenta z `if` pogoji. Spodaj je prikazano preverjanje argumentov (ang. input validation) za funkcijo `GetTagFromDatetime`. Argument je podan v obliki niza v katerem so tri besede, ločene s presledkom. Funkcijo bi se lahko naredilo tudi tako, da sprejme več argumentov, a se končni rezultat ne bi razlikoval:

```
public string GetTagFromDatetime(string s1)
{
    string[] input = s1.Split(' ');
    if (input.Length != 3){
        return "Napačen format vnosa. Pravilen vnos je yyyymm"+
        "ddhh:mm:ss tag";}
    if (input[0].Length != 8){
        return "Napačen format vnosa. Pravilen vnos je yyyymm"+
        "dd";}
    foreach (char c in input[0]){
        if (c < '0' || c > '9'){
            return "Datum mora vsebovati le številke";}
    }
    if (!input[0].StartsWith("20")) {
        return "Datum mora začeti z letnico 20**"; }
    if ( (input[1].Length != 8) || (input[1][2] != ':')
    || (input[1][5] != ':') ) {
        return "Napačen format časa. Pravilen: hh:mm:ss";}
    foreach (char c in input[1]) {
        if ((c < '0' || c > '9') && (c != ':')) {
```

```
            return "Napačen format časa. Pravilen: hh:mm:+
            "ss";}
    }
    if (! (input[2].Equals("temperature") || input[2].Equals
    ("humidity") || input[2].Equals("active_pwr") ) ) {
        return "Napačen format tag-a. Možnosti: temperature"+
        ", humidity, active_pwr";}
}
```

Spodaj je primer sestavljanja poizvedbe za MongoDB v C#:

```
IMongoQuery nowTime = Query<MongoM2MGeneric>.GT(g
=> g.Time, todayTimeLong);
IMongoQuery startTime = Query<MongoM2MGeneric>.LTE
(g => g.Time, nowTimeLong);
IMongoQuery combinedTime = Query.And(nowTime, star
tTime);
MongoCursor<MongoM2MGeneric> allDocs = data.Find(c
ombinedTime);
```

Spodaj je prikazana obravnava izjem v spletni storitvi. Do izjeme `AbandonedMutexException` pride, ko nit naleti na zaklep, ki ga neka druga nit ni sprostila. `MongoConnectionException` kadar MongoDB-jevi gonilniki ne uspejo vzpostaviti povezave z podatkovno bazo [24].

```
catch(AbandonedMutexException ame)
{
    return new string[]{"Zaklep je še v veljavi" };
}
catch(MongoDB.Driver.MongoConnectionException m)
{
    return new string[]{"Napaka pri povezavi z MongoDB: "+m.me
    ssage};
```

```
}
finally
{
    mut.releaseMutex();
}
```

4.6 Namestitev, testiranje in analiza rezultatov

4.6.1 Namestitev

Prvi cilj je bila izdelava spletne storitve v C# z uporabo .NET framework. Sledila je namestitev MongoDB gonilnikov. Še prej je bilo treba spoznati, kako se poganja osnovne ukaze z MongoDB. Potrebno je bilo raziskati, kje v podatkovni bazi so shranjeni podatki z napravic. Za to opravilo smo si delo olajšali s programom MongoVUE, s katerim se hitreje raziskuje vnose v podatkovni bazi, predvsem pa je delo z njim bolj pregledno. Spoznali smo, da se zapisi iz naprav hranijo v zbirki `m2mgeneric`, znotraj podatkovne baze, imenovane `ace`; v zbirki `event` pa so shranjeni dogodki, kjer se ustvarijo zapisi v primeru alarmnih vrednosti naprav. V približno enem letu občasnega delovanja naprav se je v zbirki `m2mgeneric` nabralo 1.187.000 vnosov, v `event` pa 361.

4.6.2 Testiranje

Testiranje spletne storitve smo opravili s prej omenjeno *mFi* opremo. Za pravilno delovanje funkcij smo klicali funkcije iz spletnega brskalnika. Že po prvih poskusih je bilo jasno, da funkcije potrebujejo vsaj nekaj sekund za odgovor. Zato smo omejili globino branja, kar pa ni kaj prida doprineslo k odzivnosti (za merjenje časa smo uporabili razred `StopWatch`). Mogoče bi se omejitev globine branja bolje izkazala pri količinsko večjih zbirkah. Pri tem

nam je pomagala narava MongoDB-ja, da zapisuje vnose po časovnem redu. S pomočjo funkcij razreda `StopWatch` smo prepoznali, kateri deli funkcije so najbolj časovno potratni. Izkazalo se je, da se večino časa zapravi za nalaganje elementov iz zbirke v strukturo tipa `List<>` (primer spodaj).

```
List<MongoM2MGeneric> reducedList = allDocs.ToList();
```

Pri iskanju rešitve smo spoznali, da je za delo z MongoDB v programskem jeziku C# manj virov kot za delo z MongoDB iz njegove ukazne lupine. Branje iz MongoDB se začne pri najstarejših vnosih, za hiter odziv nekaterih funkcij pa je ključnega pomena, da se branje začne pri najnovejših vnosih. Funkcija `Reverse()` ni obrodila sadov. Z funkcijo `SetSortOrder()` sicer bi lahko dosegli, da bi rezultati branja bili v obratnem vrstnem redu, vendar nam to ne bi koristilo pri odzivnosti. Rešitev smo našli v funkciji `SetSkip()`, s katero preskočimo določeno število vnosov (prikazano spodaj).

```
allDocs.SetSkip(count - search_depth);
```

Na tak način se je na primer trajanje funkcije `ReadLastValuesThread` skrajšalo iz 14 sekund na 1,4 sekund. S preverjanjem posredovanih argumentov funkcijam smo dosegli, da funkcije ne sprožijo izjeme ob prejetih napačnih podatkih. Občasno spletna storitev ni bila dosegljiva preko spletnega brskalnika zaradi napake *The Web server is configured to not list the contents of this directory*, ampak je bil to problem le začasne narave. Med tem časom je spletna storitev bila še vedno dosegljiva iz uporabniške aplikacije.
Do nepričakovane nevšečnosti je prišlo, ko se je v Sloveniji spremenila ura na zimski čas. Visual Studio pri uporabi objektov tipa `DateTime` samodejno dela še pretvorbo glede na trenutni čas za varčevanje s svetlobo (ang. daylight saving). Pri tem včasih pride do ene ure zamika, na primer uro 15.00 spremeni v 16.00. Ker se nam je ta zadeva zdela sekundarnega pomena, nismo spreminjali delovanja časa. Pretvorbo časa bi se dalo narediti ročno,

mogoče obstajajo tudi funkcije za to opravilo, v končni fazi pa si lahko ta čas po želji prilagaja sam uporabnik spletne storitve.

4.6.3 Analiza

Slabosti, ki naša spletna storitev prinaša, so:

- možnost napada na strežnik, zaradi katerega bi lahko spletna storitev prenehala delovati, na primer DDos (napad za zavrnitev storitve),
- strežnik, na katerega so *mFi* naprave povezane, mora imeti nameščen operacijski sistem *Windows*. Obstaja možnost, da bi naša spletna storitev delovala na *Linux* operacijskem sistemu z uporabo *Mono* ogrodja (ang. framework), ampak te možnosti v našem diplomskem delu nismo raziskovali. Brez uporabe naše spletne storitve, se *mFi* naprave lahko povežejo tudi na *pametni* telefon, če je ta dovolj zmogljiv,
- zmanjšana zanesljivost delovanja, ker spletna storitev deluje kot dodaten člen med *mFi* napravami in uporabnikom,
- potencialno slabša odzivnost, zaradi uporabe počasne internetne povezave do spletne storitve,
- časovno potratno branje iz podatkovne baze, če se išče stare podatke (spletna storitev začne branje pri najbolj svežih podatkih).

Z izdelano rešitvijo smo dosegli funkcionalen način, kako dostopati do podatkov naprav iz bilokaterega kraja, kjer imamo dostop do internetne povezave. Uporabniška aplikacija nam nudi možnost obveščanja z zvočnim signalom, spletna storitev pa nam nudi neomejen dostop do podatkov naprav.

Poglavje 5

Sklepne ugotovitve

Idejo za projekt smo dobili ob spoznanju, da mnogo uporabnikov pogreša način za standardiziran dostop do podatkov teh naprav. Po prvih negotovih korakih, kjer še nismo imeli točne predstave, kako bi rešitev zgledala, smo preučili kaj je bistvo spletne storitve ter ostalih tehnologij, ki smo jih uporabili.

Na voljo smo imeli dve napravi: senzor temperature in vlage ter *pametno* vtičnico. Razvili smo spletno storitev, ki z uporabo protokola SOAP nudi informacije o teh napravah. V analizi smo pokazali, da je izdelana rešitev funkcionalna, pri čemer imamo v mislih predvsem navezo spletne storitve z okenskim programom v *WPF*.

Med možnostmi za izboljšavo bi bila tudi izdelava novih funkcij v spletni storitvi. Ta opcija trenutno ni potrebna, bi pa lahko bila dobrodošla, če bi se uporabljale nove naprave. Manjka tudi možnost, da bi lahko uporabniški aplikaciji naknadno dodali povezavo do spletne storitve, ampak ker se strežnik načeloma ne premika, nas ta pomanjkljivost ne moti. Spletna storitev bi lahko bila popolnejša in obsežnejša z uporabo večjega števila *mFi* naprav.

Slike

Seznam kratic in simbolov

BSON (ang.) *Binary JavaScript Object Notation*: binarni format za zapis podatkov, ki ga uporablja MongoDB.
HTML (ang.) *Hyper Text Markup Language*: označevalni jezik za izdelavo spletnih strani.
HTTP (ang.) *Hyper Text Transfer Protocol*: protokol za prenos informacij po spletu.
JSON (ang.) *JavaScript Object Notation*: format za zapis podatkov.
NoSQL (ang.) *Not Only Structured Query Language*: družina podatkovnih baz, ki ne temeljijo na SQL programskem jeziku.
RDBMS (ang.) *Relational DataBase Management System*: sistem za upravljanje z relacijskimi bazami.
RPC (ang.) *Remote Procedure Call*: dostop do procedure/funkcije preko računalniškega omrežja.
SMTP (ang.) *Simple Mail Transfer Protocol*: protokol za prenos elektronske pošte **TCP** (ang.) *Transmission Control Protocol*: protokol za nadzor prenosa preko računalniškega omrežja.
URL (ang.) *Uniform Resource Locator*: enolični naslov spletnega mesta.
UTC (ang.) *Coordinated Universal Time*: mednarodno sprejet univerzalni čas.
UTF-8 (ang.) *Unicode Transformation Format 8*: način kodiranja unicode znakov.
UTP (ang.) *Unshielded Twisted Pair*: tip kabla, ki se uporablja pri telekomunikacijah, predvsem v računalniških ethernet omrežjih.

WPF (ang.) *Windows Presentation Foundation*: ena izmed Microsoftovih tehnologij za izdelavo grafičnih uporabniških vmesnikov.
WSDL (ang.) *Web Service Description Language*: označevalni jezik za spletne storitve.
XML (ang.) *eXtensible Markup Language*: označevalni jezik, ki se uporablja predvsem za prenos podatkov.

Literatura

[1] Ubiquiti Networks, dostopno na: `http://www.ubnt.com`

[2] Uradna stran Ubiquiti Networks Inc., dostopno na: `http://www.ubnt.com/enterprise/mfi`

[3] J. Snell. *O'Really Programming webservices with Soap*, O'Really Media, 2001.

[4] R. Erglander. *Java and SOAP*, O'Really Media, 2002, poglavje 2.

[5] w3c, *intro*, dostopno na: `http://www.w3.org/TR/soap12-part1/intro`

[6] Understanding SOAP, MSDN Microsoft, dostopno na: `http://msdn.microsoft.com/en-us/library/ms995800.aspx`

[7] E. Cerami. *Web Services Essentials*, O'Really Media, 2002.

[8] w3c, *xml*, dostopno na: `http://www.w3.org/XML`

[9] J. Kao. *Developer's Guide to Building XML-based Web Services*, The Middleware Company, 2001, stran 9.

[10] Wikipedia, *Document-oriented database*, dostopno na: `http://en.wikipedia.org/wiki/Document-oriented database`

[11] mongodb.org, *BSON*, dostopno na: `http://docs.mongodb.org/meta-driver/latest/legacy/bson`

[12] mongodb.org, *manual*, dostopno na:
`http://docs.mongodb.org/manual/MongoDB-manual.pdf`

[13] E. Plugge, P. Membrey, T. Hawkins. *The Definitive Guide to MongoDB*, Apress, 2010, poglavje 3, stran 41.

[14] JSON, dostopno na: `http://www.json.org`

[15] CouchDB, dostopno na: http://wiki.apache.org/couchdb/

[16] mongodb.org, *blog*, dostopno na:
`http://blog.mongodb.org/post/114440717/bson`

[17] bsonspec.org, dostopno na: `http://bsonspec.org`

[18] Wikipedia, *Comparison of data serialization formats*, dostopno na:
`http://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats`

[19] students.science.uu.nl. *bson*, dostopno na:
`http://www.students.science.uu.nl/ 3902803/bson.html`

[20] Oracle Java documentation, *wsimport*, dostopno na:
`http://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html`

[21] K. Chodorow. *MongoDB: The Definitive Guide*, O'Really Media, 2013.

[22] K. Banker. *MongoDB in Action*. Manning Publications Co., 2012, poglavje 2.

[23] Mutex, MSDN Microsoft, dostopno na:
`http://msdn.microsoft.com/en-us/library/system.threading.mutex(v=vs.110).aspx`

[24] The MongoConnectionException class, dostopno na:
`http://php.net/manual/en/class.mongoconnectionexception.php`